Approximate Policy Iteration for Generalized Semi-Markov Decision Processes: an Improved Algorithm

Emmanuel Rachelson[1], Patrick Fabiani[1], and Frédérick Garcia[2]

[1] ONERA-DCSD
2, avenue Edouard Belin, F-31055 Toulouse, FRANCE
emmanuel.rachelson, patrick.fabiani@onera.fr,
[2] INRA-BIA
Chemin de Borde Rouge, F-31326 Castanet-Tolosan, FRANCE
fgarcia@toulouse.inra.fr

Abstract. In the context of time-dependent problems of planning under uncertainty, most of the problem's complexity comes from the concurrent interaction of simultaneous processes. Generalized Semi-Markov Decision Processes represent an efficient formalism to capture both concurrency of events and actions and uncertainty. We introduce GSMDP with observable time and hybrid state space and present an new algorithm based on Approximate Policy Iteration to generate efficient policies. This algorithm relies on simulation-based exploration and makes use of SVM regression. We experimentally illustrate the strengths and weaknesses of this algorithm and propose an improved version based on the weaknesses highlighted by the experiments.

1 Introduction

Generalized Semi-Markov Decision Processes (GSMDP) [18] are an efficient and elegant formalism to describe planning problems that present the three features of decision under uncertainty, continuous time and concurrent activities. Imagine, for instance, having to plan the exploitation of a subway network, where available actions only consist in introducing or removing trains from service. In this problem, the goal is to maximize the number of passengers going through the network while minimizing the exploitation cost of the subway. Passenger arrival times, movements going in and out of the trains and possible delays in the system make the outcome of every action uncertain with regard to the next state and the date of the next decision epoch. On top of that, the flow of passengers and their destinations depend greatly on the time of day. Another problem that combines the difficulties of planning under uncertainty, hard temporal constraints and concurrent events and actions is airplanes taxiing management. Finally, coordinating an autonomous UAV's plan with regard to its environment is easily described as a planning problem where the dynamics result from the concurrent interaction of exogenous processes (other agents, environment evolution, mission change, ...). All these processes can be easily captured as GSMDP.

We present the GSMDP formalism in section 2 and illustrate how we include time as an observable variable in order to deal with time dependent problems. We also put GSMDP in perspective with standard Markov Decision Processes (MDP), approaches for dealing with continuous variables and modelling approaches for concurrent stochastic processes. Then, in section 3, we present the simulation-based reinforcement learning algorithm we developped in order to deal with the inherent difficulties of planning in GSMDP. Section 4 focuses on the evaluation of this algorithm and highlights its main weakness. Finally we introduce an improved version of the algorithm in section 5.

2 Generalized Semi-Markov Decision Processes

2.1 Time, concurrency and uncertainty

MDP [12] have become a popular model for describing problems of planning under uncertainty. Formally, an MDP is composed of a 4-tuple $< S, A, P, r >$, where S is a countable set of states for the system, A is the countable set of possible actions, $P(s'|s,a)$ is a probability distribution function providing the transition model between states and $r(s,a)$ is a reward value associated with the (s,a) transition, used to build criteria and to evaluate actions and policies. Solutions to MDP problems are often given as *Markovian policies* π, namely functions that map current states to actions. One can introduce criteria to evaluate these policies, as the discounted reward criterion given in equation 1. Criteria permit definition of the *value function* V^π associated with a policy. An important result concerning MDP is that for any history-dependent policy, there exists a Markovian policy which is at least as good with regard to a given criterion. Consequently, one can safely search for optimal control policies in the restricted space of Markovian policies without loss in optimality. Finally, algorithms as *value iteration* or *policy iteration* rely on the fact that the optimal policy's value function V^* obeys Bellman's optimality equation (equation 2) [1].

$$V_\gamma^\pi(s) = E\left(\sum_{\delta=0}^{\infty} \gamma^\delta r(s_\delta, \pi(s_\delta)\right) \tag{1}$$

$$V^*(s) = \max_{a \in A}\left[r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a)V^*(s')\right] \tag{2}$$

Among the different approaches designed to include time in the MDP framework, one could mention Semi-MDP [12], Time-dependent MDP [2] and more recently XMDP [15]. Additionnaly, dealing with observable time can take advantage of algorithms designed to manage continuous or hybrid state spaces as in [7] or [5] for instance. For a more detailed overview of the relationship between time and MDP, we refer the reader to [14]. All these models rely on the introduction of a continuous variable in the model in order to incorpor ate the time dependency into the transition function. However, writing transition and duration functions for such models is often a very complex task and requires a lot

of engineering. For instance, the effect of a *RemoveTrain* action on the global state of the subway problem is the result of several concurrent processes: passenger arrivals, trains movements, removal of one train, etc.; all compete to change the system's state and there is no simple way to summarize all these processes' concurrent stochastic influence into the transition and duration functions.

In the stochastic processes litterature, concurrent Markov processes are modelled as Generalized Semi-Markov Processes (GSMP) [6]. A GSMP is a natural representation of several concurrent SMP affecting the same state space. [18] introduced Generalized Semi-Markov Decision Processes (GSMDP) in order to model the problem of decision under uncertainty where actions compete with concurrent uncontrolable stochastic events. A GSMDP describes a problem by factoring the global transition function of the process by the different stochastic contributions of concurrent events. This makes GSMDP an elegant and efficient way of describing the complexity of time-dependent stochastic problems. We introduce the formal definition of GSMDP in the next section and focus on GSMDP with continuous observable time in the rest of the paper.

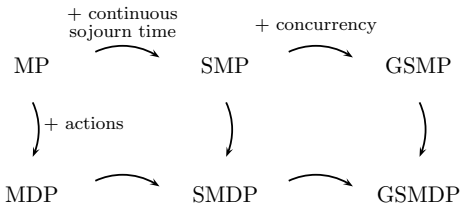


Fig. 1. From MP to GSMDP

2.2 GSMDP

We start from the stochastic process point of view, with no decision making. Formally, a GSMP [6] is described by a set S of states and a set E of events. At any time, the process is in a state s and there exists a subset E_s of events that are called *active* or *enabled*. These events represent the different concurrent processes that compete for the next transition. To each active event e, we associate a clock c_e representing the duration before this event triggers a transition. This duration would be the sojourn time in state s if event e was the only active event. The event e^* with the smallest clock c_{e^*} (the first to trigger) is the one that takes the process to a new state. The transition is then described by the transition model of the triggering event: the next state s' is picked according to the probability distribution $P_{e^*}(s'|s)$. In the new state s', events that are not in $E_{s'}$ are disabled (which actually implies setting their clocks to $+\infty$). For the events of $E_{s'}$, clocks are updated the following way:

- If $e \in E_s \setminus \{e^*\}$, then $c_e \leftarrow c_e - c_{e^*}$
- If $e \notin E_s$ or if $e = e^*$, pick c_e according to $F_e(\tau|s')$

The first active event to trigger then takes the process to a new state where the above operations are repeated. One first important remark concerning GSMP

is that the overall process does not retain Markov's property anymore: knowing the current state s is not sufficient to predict the distribution on the next state of the process. [10] showed that by augmenting the state space with the events' clocks, one could retain the Semi-Markov behaviour for a GSMP, we will discuss this issue in the next section. Introducing action choice in a GSMP yields a GSMDP as defined by [18]. In a GSMDP, we identify a subset A of controlable events or actions, the remaining ones are called uncontrolable or exogeneous events. Actions can be enabled or disabled at will and the subset $A_s = A \cap E_s$ of activable actions is never empty since it always contains at least the "idle" action a_∞ (whose clock is always set to $+\infty$) which, in fact, does nothing and lets the first exogeneous event take the process to a new state. As in the MDP case, searching for control strategies on GSMDP implies defining rewards $r(s, e)$ or $r(s, e, s')$ associated to transitions and introducing policies and criteria.

2.3 Controlling GSMDP

As mentionned before, the transition function for the global semi-Markov process does not retain the Markov property without augmenting the state space. In the classical MDP framework, one can make use of the Markov property of the transition function to prove that there exists a Markovian policy (depending only on the current state) which is at least as good as any history-dependent policy [12]. In the GSMDP case however, this is no longer possible and in order to define criteria and to find optimal policies, we need - in the general case - to allow the policy to depend on the whole *execution path* of the process. An execution path [18] of length n from state s_0 to state s_n is a sequence $\sigma = (s_0, t_0, e_0, s_1, \ldots, s_{n-1}, t_{n-1}, e_{n-1}, s_n)$ where t_i is the sojourn time in state s_i before event e_i triggers. As in [18], we define the discounted value of σ by:

$$V_\gamma^\pi(\sigma) = \sum_{i=0}^{n-1} \gamma^{T_i} \left(\gamma^{t_i} k(s_i, e_i, s_{i+1}) + \int_0^{t_i} \gamma^t c(s_i, e_i) dt \right) \tag{3}$$

where k and c are traditional SMDP lump sum reward and reward rate functions, and $T_i = \sum_{j=0}^{i-1} t_j$. One can then define the expected value of policy π in state s as the expectation over all execution paths starting in s: $V_\gamma^\pi(s) = E_s^\pi \left[V_\gamma^\pi(\sigma) \right]$.

This provides a criterion for evaluating policies. The goal is now to find policies that maximize this criterion. The main problem here is that it is hard to search the space of history-dependent policies. On the other hand, the supplementary variable technique is often used to transform non-Markovian processes into Markovian ones. It consists in augmenting the state space with just enough variables so that the distribution over future states only depends on the current values of these variables. In [10], Nielsen augments the natural state s of the process with all the clock readings and shows that this operation brings Markov behavior back to the GSMP process. We will note this augmented state space (s, c) for convenience. Unfortunately, it is unrealistic to define policies over this augmented state space since clock readings contain information about the *future* of the system. From here, several options are possible:

– One could decide to sacrifice optimality and to search for "good" policies among a restricted set of policies, say the policies defined on the current natural state only.
– One could also search for representation hypothesis that simplify the GSMDP model and that make natural state Markovian again.
– One could compute optimal policies on the augmented state space (s, c) and then derive a policy on observable variables only.
– Finally, one could look for a set of *observable* variables which retain Markov's property for the process, for example the set composed of the natural state of the process s, the duration for which each active event e_i has been active τ_i and its activation state s_i. We will note this augmented state (s, τ, s_a)

[18] is based on the second option listed above. In the next section, we introduce a new reinforcement learning method based on simulation-guided approximate policy iteration, designed to deal with large state spaces for GSMDP with continuous observable time and that can be adapted to the three other options. For a more detailed discussion about the motivations of this approach, see [14].

3 Simulation-based Approximate Policy Iteration for GSMDP

3.1 Approximate Policy Iteration

Our algorithm belongs to the Approximate Policy Iteration (API) family of algorithms. Policy Iteration is an algorithm for solving MDP which searches the policy space in a two-step fashion as illustrated in equation 4. Given a policy π_n at step n, the first step consists in computing the value of π_n. The second step then performs a Bellman backup in every state of the state space, thus improving the policy. An important property of policy iteration is its good anytime behaviour: at any step n, policy π_n will be at least as good as any previous policy. Policy Iteration usually converges in less iterations than the standard Value Iteration algorithm but takes longer since the evaluation step is very time consuming. To deal with real problems, one needs to allow for approximate policy evaluation (as in [8]) since exact computation is often infeasible. There are few theoretical guarantees on convergence and optimality of API, as explained in [9].

$$\pi_{n+1}(s) = \underset{a \in A}{argmax} \left[r(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V^{\pi_n}(s') \right] \tag{4}$$

The baseline of our algorithm is to start with an initial policy defined over what we chose as the observable state space for a GSMDP and incrementaly improve its quality on the states that are *relevant* to the current problem.

3.2 Algorithm

Partial state space exploration. We deal with continuous - possibly hybrid - state spaces, therefore, unless we make strong hypothesis of the shape of the

transition and reward functions (as in [2] or [18]) it is not possible to compute the exact value of policy π_n. Instead, we decide to sample the state space in order to obtain an evaluation of V^{π_n}. We perform N_0 simulations of the current policy π_n starting from the current state of the process and we store the triplets of states, times and rewards $(s_\delta, t_\delta, r_\delta)$ obtained. Thus, by summing all the rewards from one state to the end of the simulation, one execution path σ_i yields a value function V_i over the discrete set of states explored during simulation (equation 3). All the value functions issued from simulation yield a training set $\{(s, v)\}$, $s \in S, v \in \mathbb{R}$, from which we wish to generalize a value function $\tilde{V}$ over all states. It is important to note that since time is a state variable and because of the *causality principle*, the only possible loops in the state dynamics are instantaneous loops (the process doesn't go back in time) and there is a zero probability of an infinite sequence of instantaneous loops. Consequently, simulations come to an end when they reach the absorbing states at $t = horizon$. As $N_0 \to \infty$, the subspace explored by the simulations tends to the reachable subspace from s_0 and each state is visited with a probability corresponding to the stationary Markov chain defined by (s_0, c_0, π_n) [10]. Therefore:

$$\forall s \in Reachable(s_0), \lim_{N_0 \to \infty} \frac{\sum_{i=1}^{N_0} V_i(s)}{N_0} = V^{\pi_n}(s) \tag{5}$$

We can summarize this approach by saying that we let the current policy guide our sampling in the state space in order to reach a good approximation of the curent policy's value function over all reachable states.

Value function generalization. Unfortunately, we cannot perform an infinite number of simulations and we are left with a finite set of samples for the value function evaluation. Consequently, we wish to generalize this information in order to interpolate the value function over all states. We use th samples as a training set for a regression method that will generalize it to the entire state space. Several approaches to regression based reinforcement learning have been proposed in the machine learning community - methods based on trees [4], evolutionary functions [17], kernel methods [11], etc. - but few have been coupled with policy simulation. We chose to focus on support vector regression (SVR) because of its ability to handle the large dimension spaces over which our samples are defined. SVR belong to the family of kernel methods and can be used for both regression and classification (SVC). Training a standard SVR over a given training set corresponds to looking for a hyperplane interpolating the samples in a higher dimensional space called *feature space*. Practically, SVR take advantage of the *kernel trick* to avoid expressing the feature space explicitely. For more details on SVR and SVC, we refer the reader to [16]. In our case, we call $\tilde{V}_n(s)$ the interpolated value function of policy π_n.

Online policy instanciation. Finally, while simulation-based exploration and SVR generalization of the value function are techniques dedicated to improve

the evaluation step of approximate policy iteration, the third specificity of our algorithm deals with improving the optimization step. For large and possibly continuous state spaces, it might be very long or impracticable to compute the one-step improvement of the policy. Indeed, most of the time, computing a complete policy is irrelevant since most of this policy will never be used for the simulation-based evaluation step. This idea is the basis of *asynchronous* policy iteration. Instead, it might be easier to compute online the one-step lookahead best action in the current state with respect to the stored value function. More precisely, in a standard MDP, the optimization step consists in solving equation 6 in every state:

$$\pi_{n+1}(s) \leftarrow \arg\max_{a \in A} \tilde{Q}_{n+1}(s,a) \tag{6}$$
$$\text{with: } \tilde{Q}_{n+1}(s,a) = r(s,a) + \sum_{s' \in S} P(s'|s,a)\tilde{V}_n(s,a)$$

For continuous state spaces, computing π_{n+1} implies being able to compute integrals over P and $\tilde{V}_n$. We wish not make hypothesis on the model used and therefore will perform a discretization for evaluation of the integral. Finally, since the model of P is not necessarily known to the decision maker and since we have a simulator of our system, we will make a second use of this simulator for the purpose of evaluating the expected reward $\tilde{Q}_{n+1}(s,a)$ associated with performing action a in state s with respect to value function $\tilde{V}_n$ (equation 7). At the end of the evaluation phase, the value function $\tilde{V}_n$ is stored and no policy is computed from it. Instead, we immediately enter a new simulation phase but whenever the policy π_{n+1} is asked for the action to perform in the current state s it performs *online* the estimation of all Q-values for state s and then choses the best action to perform. The speed up in the execution of the policy iteration algorithm is easy to illustrate for discrete state space problems since we replace $|S|$ evaluations of the Q-values for policy update by the number of states visited during one simulation. This is especially interesting in the case of time dependent problems since a state is never visited twice (except for instantaneous loops). Consequently, $\tilde{Q}_{n+1}(s,a)$ is calculated by simply simulating N_a times the application of a in s and observing the set of $\{(r_i, s'_i)\}$ as in equation 7. Then the policy returns the action which corresponds to the largest Q-value. We call this online instanciation of the policy "online approximate policy iteration".

$$\tilde{Q}_{n+1}(s,a) = \frac{1}{N_a} \sum_{i=1}^{N_a} \left[r_i + \tilde{V}_n(s'_i) \right] \tag{7}$$

To summarize, the improvement of the current policy is performed online: for each visited state (starting in s_0) we perform one Bellman backup using the value function evaluation from the previous step, then apply the best action found to move on to the next state. This can also be seen as letting the policy guide the choice for the subset of states over which we improve the policy at step $n+1$.

Algorithm 1 Online-ATPI

main:
Input : π_0 or $\tilde{V}_0$, s_0
loop
  $TrainingSet \leftarrow \emptyset$
  for $i = 1$ to N_0 **do**
    $\{(s, v)\} \leftarrow$ simulate$(\tilde{V}, s_0)$
    $TrainingSet \leftarrow TrainingSet \cup \{(s, v)\}$
  end for
  $\tilde{V} \leftarrow$ TrainApproximator$(TrainingSet)$
end loop

simulate$(\tilde{V}, s_0)$**:**
$ExecutionPath \leftarrow \emptyset$
$s \leftarrow s_0$
while horizon not reached **do**
  $action \leftarrow$ ComputePolicy$(s, \tilde{V})$
  $(s', r) \leftarrow$ GSMDPstep$(s, action)$
  $ExecutionPath \leftarrow ExecutionPath \cup (s', r)$
end while
convert execution path to value function $\{(s, v)\}$ (eqn 3)
return $\{(s, v)\}$

ComputePolicy$(s, \tilde{V})$**:**
for $a \in A$ **do**
  $\tilde{Q}(s, a) = 0$
  for $j = 1$ to N_a **do**
    $(s', r) \leftarrow$ GSMDPstep(s, a)
    $\tilde{Q}(s, a) \leftarrow \tilde{Q}(s, a) + r + \gamma^{t'-t}\tilde{V}(s')$
  end for
  $\tilde{Q}(s, a) \leftarrow \frac{1}{N_a}\tilde{Q}(s, a)$
end for
$action \leftarrow \arg\max_{a \in A} \tilde{Q}(s, a)$
return *action*

Algorithm overview and implementation. The *online Approximate Temporal Policy Iteration* (online-ATPI) algorithm is presented in algorithm 1.

Note that in algorithm 1, s actually denotes the part of the state that is observable to the policy. This makes online-ATPI adaptable to any of the sets of policy variables presented in section 2.3. We implemented a version of online-ATPI on the natural state of the process. It will be important, in future work, to test the algorithm on the (s, c) and (s, τ, s_a) Markovian states. In the current version of the algorithm, the generated value function is usable "as is" in order to derive a greedy policy for the problem. In the case of the (s, c) state space, one more step is necessary in order to build an estimator of the current clocks to make such a policy usable.

4 Evaluation

4.1 The subway control problem

We tested online-ATPI on a simple instance of the subway problem which had 4 trains and 6 stations. In this problem, the goal is to optimize the exploitation cost of the subway over a whole day. Events such as $trainMovement$ and actions such as $addTrain$ cost money (for the energy consumption) while every passenger exiting the subway brings positive rewards (ticket price). The problem was formalized using the following concurrent events (which can be disabled in non-relevant states):

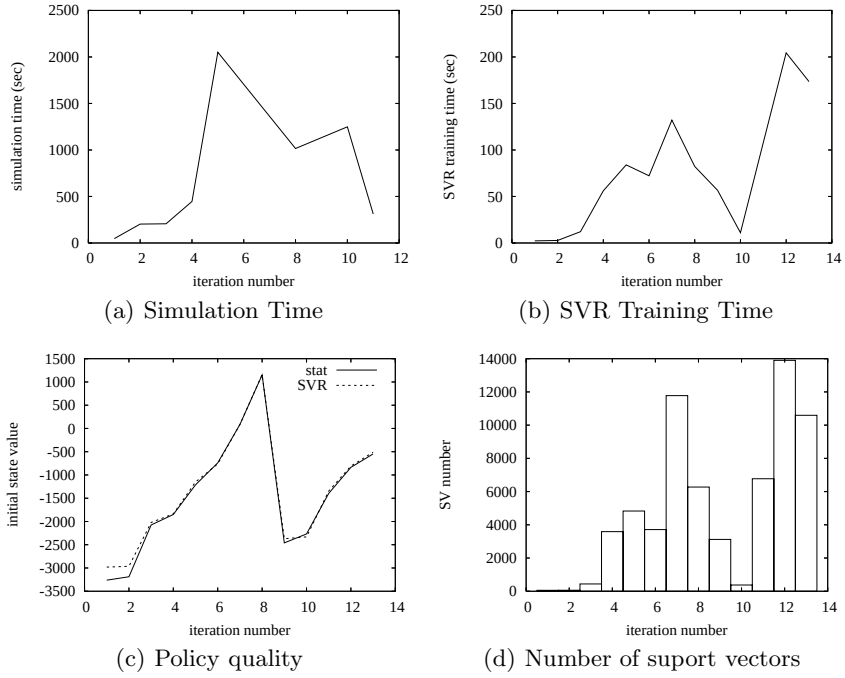
- $passengerArrival_i$: passenger arrival in station i - exogeneous.
- $trainMovement_j$: movement of train number j, only enabled if the train is in service. This event actually also includes passengers going in and out of the train - exogeneous.
- $addTrain_j$: adds train j in station 0 - controlable.
- $removeTrain_j$: removes train j from service, only enabled if train j is in station 0 - controlable.
- a_∞: NoOp - controlable.

The state space included continous (time), discrete (train position), and boolean (redundant variables as $NoTrainAtStation_j$) variables. Finally the problem had 19 concurrent events and a hybrid state space of dimension 22. Depending on the time of day, the transition functions for each event described the continuous evolution of passenger flows, destinations, arrival frequencies, etc.

This series of experiments used the VLE multimodel simulation engine [13], based on the DEVS [19] formalism. We developped GSMP and GSMDP extensions to the VLE platform and used them for all simulation tasks. This allowed us to take advantage of the interoperability of DEVS models. More specifically, it allows for simulation of coupled models, *ie.* if an event or a whole system interacts with our GSMDP but is not written as a GSMP itself, then DEVS coupling would still allow for simulation of the whole system, retaining the properties of event-driven discrete event simulation. This does not have an immediate impact on the current algorithm but extends its application scope to discrete events dynamic systems. We used $N_0 = 20$ simulations per policy iteration and $N_a = 15$ samples per action (Cf. algorithm 1). The SVR was a standard ϵ-SVR, adapted from the LIBSVM library [3], with Gaussian kernels and $\sigma^2 = 20$. After N_0 simulations, the training set for the SVR had around 45000 points. The experiments were ran on a 1.7 Ghz single core processor with 884Mio of RAM.

4.2 Discussion

Figures 2(a) to 2(d) present the simulation results when the algorithm is initialized with a default policy that sets all the trains to run all day long. Figure 2(a) and 2(d) illustrate the fact that the number of support vectors in the SVR is the factor that greatly handicaps the simulation time. This could be solved by using



(a) Simulation Time (b) SVR Training Time

(c) Policy quality (d) Number of suport vectors

other SVR techniques as ν-SVR. However, since N_0 simulations always generate approximately the same number of training samples, the number of support vectors we have to store is bounded. The most interesting features highlighted by these experiments come from in figure 2(c).

First of all, between iteration 1 and 8, the expected value for the initial state rises, as one could expect from a "policy iteration"-like algorithm. As a matter of fact, this increase is not necessarily linear and depends on the problem's structure. If the policy takes the simulation to states that are "far" from explored states (states for which the interpolated value might be erroneous) and that provide very bad rewards, it can happen that the initial state's expected value drops for one iteration. This is the drawback from partial exploration of the state space and interpolation: very good or very bad regions of the state space might be discovered late in the iterations (or might not be discovered at all).

Secondly, one can notice that online-ATPI quickly (in less than ten iterations) finds policies with positive expected values. This actually means that - with the hard economical constraints we put on the problem - it was still possible to find a policy that makes the subway profitable while the best "hand-made" policy we tested so far had an expected value of zero for s_0.

Finally, this same result illustrates the main weakness of this first version of the algorithm. After finding a good policy π_8, the initial state's expected value drops dramatically at the next iteration which is in contradiction with the monotonicity of the value function's expected evolution. This phenomenon is

due to the fact that outside the region explored at the last iteration (outside of the region where samples are dense), the SVR over-estimates the expected gain because it does not have any training samples in these regions (because they are indeed bad regions for expected values, the good policy avoided them and we did not keep track of these regions). Consequently, during the next improvement step, whenever the algorithm asks for the value of a state which is outside of this "confidence region", the value returned by the SVR is too high and the SVR cannot act as an admissible heuristic anymore. This pulls the policy back towards these bad regions of the state space. From this new (bad) policy, the improvement process starts again and the value of s_0 rises again between iterations 9 and 14.

Section 5 shows how we modify the algorithm to deal with this problem.

5 Avoiding policy degradation

In order to keep the interest of online-ATPI and to retain the monotonicity of the value function's evolution, we need to guarantee that we will never overestimate the expected value of a state with regard to the current policy. The main idea here is to keep track of previous optimizations for the policy, even on states that are not visited anymore, and to define confidence regions for the last exploration.

5.1 Keeping track of the global policy

In order not forget previous policy optimizations, we keep a history of all optimized actions throughout the iterations. Practically, we build a classifier on the pairs (s, a) (a regressor in the case of continuous actions) which generalizes the optimized actions to their neighbourhood in the continuous state space. This raises two issues: firstly, when to replace an action by another in the classifier, secondly when to use the optimized policy (and not the initial policy). These two issues are adressed the same way: we need a function defining whether we trust our interpolation in a given state or not. In other words, we need to define a confidence function, which indicates if the classifier has had enough samples around the current state in order to learn a clever action and return a reasonable solution. This notion of confidence function is crucial to the improved version of online-ATPI, we will develop it more in the next paragraph. Finally, this is how we modify the algorithm. We maintain a database $actionDB$ of $x = (s, a)$, thus keeping track of all explored states. At iteration n, the N_0 simulation provide us with a new set of optimized actions on the traversed state space $actionDB'$. We remove any sample x from $actionDB$ such that $confidence(actionDB', x) = true$, add all samples from $actionDB'$ to $actionDB$, then train the classifier $\tilde{\pi}$ on the updated $actionDB$ database. Practically, we use standard SVC for the same reasons we used SVR for regression. During simulation, we use the confidence function again to know if the region has ever been visited: if so, we use $\tilde{\pi}$, if not, we use π_0.

5.2 When to trust the SVR evaluation of the value function

However, keeping track of previous optimizations does not protect us from overestimating the expected value in poorly explored regions. When an improvement step takes us far from previously explored states, the SVR value function outputs a value which tends to the average of the explored values. The $confidence$ function we defined in the previous paragraph can be reused here to define the confidence we have in the SVR estimation. Therefore we can determine a threshold on the confidence value under which we consider the SVR $\tilde{V}$ as non-reliable. This does not guarantee the SVR to be an admissible heuristic in the confidence region, but insures that its value is close to the real expected value of every sample state in the confidence region (within ϵ for ϵ-SVR). In order to define this confidence function, we store all the (s, v) samples from the last iteration in a $valueDB$ database. These samples correspond to the evaluation of the last policy. When the $ComputePolicy$ function requests the value of a state s', if $confidence(valueDB, s') = true$, then we use the value returned by $\tilde{V}$. Else, we run N_1 simulations from s', using $\tilde{\pi}$ (or π_0 if $\tilde{\pi}$ is not relevant) in order to enrich the $valueDB$ database and to adapt $\tilde{V}$. We stop these simulations as soon as they enter the confidence region again (this speeds up simulation time and avoids refining the database in non necessary regions). This provides a good estimation of $V(s')$, refines the global $\tilde{V}$ function and extends its confidence region.

We provide a proof sketch for the question of the value function's monotonous evolution. Suppose we are performing iteration $n+1$ from state s. We want to prove that $V^{\pi_{n+1}}(s) \geq V^{\pi_n}(s)$. For convenience we neglect the approximation error in the confidence region. If s is inside the confidence region of $valueDB$, then the result is immediate. If s is outside this region then we can decompose an execution path starting in s into the path outside e_{out} and the first state to enter the confidence region again s_{in}. We have $V^{\pi_{n+1}}(s) = V(e_{out}) + V(s_{in})$, but $V(e_{out}) = V^{\pi_n}(e_{out})$ and $V(s_{in}) \geq V^{\pi_n}(s_{in})$. Therefore $V^{\pi_{n+1}}(s) \geq V^{\pi_n}(s)$.

Practically, there are several different ways of defining the confidence function: we have turned towards local linear density estimators and one-class SVR, but previous work from the statistical learning community could probably apply here. This area is still a domain of future research. However, it is important to notice that a very selective confidence function will yield very large databases and SVM and will provide very fine-grained estimations of the value function and the policy, while softer confidence functions will reduce the number of simulations to perform but will deteriorate the estimator's quality. Therefore, this confidence function defines the granularity of our value function and policy. Lastly, we use ϵ and ν-SVR for regression, but the training set used for learning is regularly refined when we perform new simulations in an unknown state, we can thus take advantage of incremental batch SVR learning which is still a very hot topic in SVR research. This last topic is particularly important to *online-ATPI* since early experiments with the improved algorithm showed that the refining and retraining process slowed heavily the policy learning process.

The improved *online-ATPI* algorithm is presented on algorithm 2.

Algorithm 2 Modified online-ATPI

main:
Input : π_0 or $\tilde{V}_0$, s_0
loop
  $valueDB$.reset(), $newActionDB$.reset()
  for $i = 1$ to N_0 **do**
    σ.reset()
    $mainProcess.s \leftarrow s_0$
    while horizon not reached **do**
      $a =$bestAction(s), activateEvent(a)
      $(s', r) \leftarrow mainProcess$.step(), σ.add(s, a, r)
    end while
    $valueDB$.convertExecutionPathToValueFunction(σ)
    $newActionDB$.add(σ)
  end for
  $\tilde{V} \leftarrow$ TrainSVR$(valueDB)$, $\tilde{\pi} \leftarrow$ TrainSVC$(actionDB)$
end loop

bestAction(s)**:**
for $a \in A_s$ **do**
  $\tilde{Q}(s, a) = 0$
  for $j = 1$ to N_a **do**
    $\tilde{Q}(s, a) = \tilde{Q}(s, a) +$ simulateWithStop(s, a)
  end for
  $\tilde{Q}(s, a) = \frac{\tilde{Q}(s,a)}{N_a}$
  return $\arg\max_{a \in A} \tilde{Q}(s, a)$
end for

simulateWithStop(s, a)**:**
activateEvent(a), $(s', r) \leftarrow mainProcess$.clone().step()
$Q \leftarrow r$, $s \leftarrow s'$
while horizon not reached & confidence(s)=false **do**
  $a = \tilde{\pi}(s)$, activateEvent(a)
  $(s', r) \leftarrow mainProcess$.clone().step()
  $Q \leftarrow Q + r$, $s \leftarrow s'$
end while
$Q = Q + \tilde{V}(s)$
$valueDB.update()$, $\tilde{V} \leftarrow$ TrainSVR$(valueDB)$
return Q

6 Conclusion

We presented a new method for policy search in large dimension, hybrid state space GSMDP. Our method relies on partial exploration of the state space, guided by the current policy, and tries to retain the monotonicity of the value function's evolution, provided by the Policy Iteration algorithm. This results in a specific Approximate Policy Iteration algorithm which we called *online-ATPI*.

In its current version, online-ATPI makes an extensive use of SVR and SVC in order to generalize discrete samples information to the continuous or hybrid state space. It also relies on a notion of granularity in the policy search, via the use of *confidence* functions regarding the pertinence of the SVM estimators. Future work will deal with evaluation of the improved algorithm.

References

1. Bellman R. E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, New Jersey.
2. Boyan J. & Littman M. (2001). Exact solutions to time dependent MDPs. *Advances in Neural Information Processing Systems*, **13**, 1026–1032.
3. Chang C.-C. & Lin C.-J.(2001). LIBSVM: a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.
4. Ernst D., Geurts P. & Wehenkel L. (2005). Tree-based batch mode reinforcement learning. *JMLR*, **6**, 503–556.
5. Feng Z., Dearden R., Meuleau N. & Washington R. (2004). Dynamic programming for structured continuous markov decision problems. In *20th Conference on Uncertainty in AI*, p. 154–161.
6. Glynn P. (1989). A GSMP formalism for discrete event systems. *Proc. of the IEEE*, **77**.
7. Hauskrecht M. & Kveton B. (2006). Approximate linear programming for solving hybrid factord MDPs. In *9th Int. Symp. on AI and Math.*
8. Lagoudakis M. & Parr R. (2003). Least-squares policy iteration. *JMLR*, **4**, 1107–1149.
9. Munos R. (2003). Error bounds for approximate policy iteration. In *Int. Conf. on Machine Learning*.
10. Nielsen F. (1998). GMSim: a tool for compositionnal GSMP modeling. In *Winter Simulation Conference*.
11. Ormoneit D. & Sen S. (2002). Kernel-based reinforcement learning. *Machine Learning*, **49**, 161–178.
12. Puterman M. (1994). *Markov Decision Processes*. John Wiley & Sons, Inc.
13. Quesnel G., Duboz R., Ramat E. & Traore M. (2007). VLE - A Multi-Modeling and Simulation Environment. In *Proc. of Summer Simulation Conf. 07*
14. Rachelson E., Fabiani P., Garcia F. & Quesnel G.(2008). Une Approche basée sur la Simulation pour l'Optimisation des Processus Décisionnels Semi-Markoviens Généralisés (english version). In *CAp08.*
15. Rachelson E., Garcia F. & Fabiani P. (2008). Extending the Bellman equation for MDP to continuous actions and continuous time in the discounted case. In *10th Int. Symp. on AI and Math.*
16. Vapnik V., Golowich S. & Smola A. (1996). Support vector method for function approximation, regression estimation and signal processing. *Advances in Neural Information Processing Systems*, **9**, 281–287.
17. Whiteson S. & Stone P. (2006). Evolutionary function approximation for reinforcement learning. *JMLR*, **7**, 877–917.
18. Younes H. & Simmons R. (2004). Solving generalized semi-markov decision processes using continuous phase-type distributions. In *AAAI*.
19. Zeigler B. P., Kim D. & Praehofer H. (2000). *Theory of modeling and simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.